(1)

ARMSTRONG LABORATORY

AN INTEGRATED MODEL DEVELOPMENT ENVIRONMENT

Patrick K. Clark
Nelly Droznin

THE ANALYTIC SCIENCES CORPORATION (TASC)
2555 UNIVERSITY BOULEVARD
FAIRBORN, OH 45324

399113

HUMAN RESOURCES DIRECTORATE
LOGISTICS RESEARCH DIVISION

318

MAY 1992

INTERIM TECHNICAL PAPER FOR PERIOD MARCH 1990 - AUGUST 1991

AIR FORCE SYSTEMS COMMAND
WRIGHT-PATTERSON AIR FORCE BASE, OHIO 45433-6573

NOTICES

This paper has been reviewed and is approved for publication.

Bradley A Lloyd

BRADLEY A. LLOYD, 1LT, USAF
Contract Monitor

[illegible]

BERTRAM W. CREAM, Chief
Logistics Research Division

REPORT DOCUMENTATION PAGE	*Form Approved* *OMB No. 0704-0188*

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE May 1992	3. REPORT TYPE AND DATES COVERED Interim - March 1990 to August 1991

4. TITLE AND SUBTITLE An Integrated Model Development Environment	5. FUNDING NUMBERS C - F33615-90-C-0007 PE - 62205F PR - 1710 TA - 00 WU - 50
6. AUTHOR(S) Patrick K. Clark Nelly Droznin	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) The Analytic Sciences Corporation (TASC) 2555 University Boulevard Fairborn, OH 45324	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) Armstrong Laboratory Human Resources Directorate Logistics Research Division Wright-Patterson AFB, OH 45433-6573	10. SPONSORING/MONITORING AGENCY REPORT NUMBER AL-TP-1992-0030
11. SUPPLEMENTARY NOTES	
12a. DISTRIBUTION/AVAILABILITY STATEMENT Approved for public release; distribution is unlimited.	12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

The Integrated Model Development Environment (IMDE) is a project being sponsored by the Armstrong Laboratory, Logistics Research Division. The goal of the IMDE contract is to demonstrate how logistics simulation can benefit from advanced software technologies. More specifically, a prototype object-oriented modeling environment for airbase logistics modeling is being developed. The environment will support graphical programming and linking of simulation objects. The eventual goal of the IMDE related research is to provide wing and headquarter level logisticians with a robust planning and re-planning system which incorporates advanced simulation and artificial intelligence technologies. This report summarizes the motivation behind the IMDE contract, the IMDE prototype modeling process, and the issues associated with transitioning the IMDE technology.

14. SUBJECT TERMS Graphical Modeling Programming Logistics Object-Oriented Simulation			15. NUMBER OF PAGES 36
			16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT UL

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std Z39-18
298-102

GENERAL INSTRUCTIONS FOR COMPLETING SF 298

The Report Documentation Page (RDP) is used in announcing and cataloging reports. It is important that this information be consistent with the rest of the report, particularly the cover and title page. Instructions for filling in each block of the form follow. It is important to ***stay within the lines*** to meet ***optical scanning requirements***.

Block 1. Agency Use Only *(Leave blank)*.

Block 2. Report Date. Full publication date including day, month, and year, if available (e.g. 1 Jan 88). Must cite at least the year.

Block 3. Type of Report and Dates Covered. State whether report is interim, final, etc. If applicable, enter inclusive report dates (e.g. 10 Jun 87 - 30 Jun 88).

Block 4. Title and Subtitle. A title is taken from the part of the report that provides the most meaningful and complete information. When a report is prepared in more than one volume, repeat the primary title, add volume number, and include subtitle for the specific volume. On classified documents enter the title classification in parentheses.

Block 5. Funding Numbers. To include contract and grant numbers; may include program element number(s), project number(s), task number(s), and work unit number(s). Use the following labels:

C	- Contract	PR	- Project
G	- Grant	TA	- Task
PE	- Program Element	WU	- Work Unit Accession No.

Block 6. Author(s). Name(s) of person(s) responsible for writing the report, performing the research, or credited with the content of the report. If editor or compiler, this should follow the name(s).

Block 7. Performing Organization Name(s) and Address(es). Self-explanatory.

Block 8. Performing Organization Report Number. Enter the unique alphanumeric report number(s) assigned by the organization performing the report.

Block 9. Sponsoring/Monitoring Agency Name(s) and Address(es). Self-explanatory.

Block 10. Sponsoring/Monitoring Agency Report Number. *(If known)*

Block 11. Supplementary Notes. Enter information not included elsewhere such as: Prepared in cooperation with...; Trans. of...; To be published in.... When a report is revised, include a statement whether the new report supersedes or supplements the older report.

Block 12a. Distribution/Availability Statement. Denotes public availability or limitations. Cite any availability to the public. Enter additional limitations or special markings in all capitals (e.g. NOFORN, REL, ITAR).

DOD	- See DoDD 5230.24, "Distribution Statements on Technical Documents."
DOE	- See authorities.
NASA	- See Handbook NHB 2200.2.
NTIS	- Leave blank.

Block 12b. Distribution Code.

DOD	- Leave blank.
DOE	- Enter DOE distribution categories from the Standard Distribution for Unclassified Scientific and Technical Reports.
NASA	- Leave blank.
NTIS	- Leave blank.

Block 13. Abstract. Include a brief *(Maximum 200 words)* factual summary of the most significant information contained in the report.

Block 14. Subject Terms. Keywords or phrases identifying major subjects in the report.

Block 15. Number of Pages. Enter the total number of pages.

Block 16. Price Code. Enter appropriate price code *(NTIS only)*.

Blocks 17. - 19. Security Classifications. Self-explanatory. Enter U.S. Security Classification in accordance with U.S. Security Regulations (i.e., UNCLASSIFIED). If form contains classified information, stamp classification on the top and bottom of the page.

Block 20. Limitation of Abstract. This block must be completed to assign a limitation to the abstract. Enter either UL (unlimited) or SAR (same as report). An entry in this block is necessary if the abstract is to be limited. If blank, the abstract is assumed to be unlimited.

TABLE OF CONTENTS

	PAGE
I. INTRODUCTION	1
II. SYSTEM CHARACTERISTICS	2
Uncomplicated Model Construction	2
Automated Configuration Control	2
Automated Data Analysis/Report Generation	3
Reusable Models/Model Parts	3
On-Line Help	4
User Interface	5
III. SYSTEM REQUIREMENTS BASELINE	5
Tool Selection	6
Four User Levels	7
Environment Architecture	7
Preprocessor	9
Simulator	10
Postprocessor	10
Kernel Model Development Environment	11
IV. MODEL DEVELOPMENT AND MAINTENANCE	11
V. POTENTIAL DATA SOURCES AND MAINTENANCE PROCEDURES	18
Potential Data Sources	18
Data Maintenance Procedures	20
VI. SYSTEM APPLICATION CONSTRAINTS	20
VII. FUTURE DIRECTIONS	21
VIII. CONCLUSIONS	22
REFERENCES	23

LIST OF FIGURES

FIGURE	TITLE	PAGE
1	Database Relationships	8
2	Premodel Preprocessor	13
3	Premodel Class Editor	14
4	Network Editor	16
5	Project Editor	17

PREFACE

The research described in this paper was performed by The Analytic Sciences Corporation (TASC) and the U.S. Air Force Armstrong Laboratory, Logistics Research Division. The TASC effort was conducted under contract number F33615-90-C-0007. The research was designed to support the Productivity Improvements in Simulation Modeling (PRISM) project, an effort to enhance the Air Force ability to evaluate, through simulation, how differing logistical support scenarios may constrain sortie generation capability. Currently, the primary research being done under PRISM is the Integrated Model Development Environment (IMDE) contract effort.

SUMMARY

The Integrated Model Development Environment (IMDE) is a project being sponsored by the Armstrong Laboratory, Logistics Research Division. The goal of the IMDE contract is to demonstrate how logistics simulation can benefit from advanced software technologies. More specifically, a prototype object-oriented modeling environment for airbase logistics simulation is being developed. The environment will support graphical programming and linking of simulation objects. The eventual goal of the IMDE related research is to provide wing and headquarter level logisticians with a robust planning and re-planning system which incorporates advanced simulation and artificial intelligence technologies. This report summarizes the motivation behind the IMDE contract, the IMDE prototype modeling process, and the issues associated with transitioning the IMDE technology.

I. INTRODUCTION

In 1990, the Air Force Armstrong Laboratory, Logistics Research Division, awarded a contract to The Analytic Sciences Corporation (TASC) for the development of an Integrated Model Development Environment (IMDE). This project was initiated as a result of internal Air Force surveys of existing logistics simulation tools. These surveys highlighted the potential for increases in productivity achievable through the use of state-of-the-art hardware and software. The predominant discrete event simulation tools currently in use are the Logistics Composite Model (LCOM) and the Theater Simulation of Airbase Resources (TSAR). Both are powerful models of airbase logistics processes, but both also have several major deficiencies. These include lengthy model configuration times, no configuration control process, and only basic data analysis and report generation capabilities. Further, the use of both models requires extensive software and analysis skills.

The purpose of this paper is to present the results of the research at about the halfway point of the effort. A much more detailed discussion of the IMDE implementation is available in the Department of Defense Standard 2167A (DOD-STD-2167A) documentation being developed as part of the contract. This paper provides an overview of the technical effort and highlights other work which might increase productivity in the simulation of airbase logistics. Specifically, this paper discusses system characteristics, system requirements baseline and constraints, model development and maintenance procedures, potential data sources and data maintenance procedures, system application contraints, and future research directions and alternatives.

II. SYSTEM CHARACTERISTICS

During the process of reviewing deficiencies of currently used airbase logistics models, talking to users of those models, and reading the user/programming manuals, several general areas became apparent for enhancing the airbase logistics modeling process. IMDE will incorporate uncomplicated model construction, automated configuration control, automated data analysis and report generation, reusable models and model parts, and on-line help. A graphical windowing user interface will tie these together. These system characteristics are discussed in the following paragraphs.

Uncomplicated Model Construction

Logistics simulation systems in current use present a serious impediment for building models to the first-time user as well as the experienced modeler. Learning the language of TSAR or LCOM well enough to independently construct simulation models can take a newcomer over six months. Even people who have become acknowledged experts with these systems spend much of their modeling time tracking down simple but elusive problems. Many of these problems can be avoided by automating the construction process to track the more repetitive but tedious tasks, freeing the analyst to spend more time actually designing the model. The IMDE will facilitate this process by using free format guided data entry windows and by providing graphical, easy-to-use simulation construction tools.

Automated Configuration Control

Part of the complexity in doing any large simulation study has little to do with the actual complexity of the simulation domain, but instead involves keeping track of which parameterizations have been made at any given point during the course of the study, the data and reports that resulted from that set of "runs" through the simulation, and the rationale for choosing each specific set of parameters (documentation). Keeping model inputs, associated outputs, and documentation tied together is a simple task for a small set of inputs, but can quickly become a nightmare during a large study. Again, these tasks dilute the creative work of the modeler and could be handled more efficiently and accurately by a computer. IMDE will provide a large degree of configuration control by using databases for storage of models, their constituent parts, and input and output data sets.

Automated Data Analysis/Report Generation

As simulation runs are completed during the course of a study, the analysis of the raw data generated often lags the decision to perform another set of runs with different parameters. A complete analysis of the data prior to starting the second set of runs often provides more insight into the choice of new parameterizations for the second set. A large amount of the data analysis task can be handled automatically, given some standardization on the output formats of raw simulation data. Tasks like table sorting and graphing results can also be easily handed off to the computer, leaving the operator free to specify which parameters should be plotted or printed. The IMDE will include a built-in postprocessor component which will do a variety of the aforementioned tasks as well as statistical analysis tasks.

Reusable Models/Model Parts

The capability to reuse a model or parts of a model for a new study can significantly accelerate the design of a new simulation model. Currently, to reuse parts of old models, parts of files containing activity networks are copied into new files. Any new functionality is then built around that model base. The problem with this method is a lack of encapsulation of the reusable parts into stand-alone units. Encapsulation is a term used widely in the object-oriented design/programming community to mean that the source code for algorithms and for data elements that the algorithms access or change are located together. With conventional models, each time a set of networks is reused it must be "carved out" of the existing file again, and there is no guarantee that these networks will be able to function as a stand-alone entity (meaning that all the data required by the networks is also available). An object-oriented approach to the storage of independent, reusable model parts allows the data and algorithms associated with any given simulation entity to be stored textually together, allowing much easier (and more confident) reuse of a part. The algorithms are originally written with the data they operate on, so there is no need for major code surgery. Rather, specific software model parts can be accessed as separate modules as long as their interface requirements with other modules are met. These modules are known as "classes" in the terminology of object-oriented programming. "Objects" are typically defined as "instances" of classes. For example, in an airbase simulation, "aircraft" may be a class which is used as a template to create some number of objects that will actually function as modeled aircraft during the simulation. (Unfortunately, terminology is not universal. Modsim II, an object-oriented simulation language, refers to classes as objects. Within this paper, the definitions given above will be used.)

An analogy of the reusability concept was drawn by Cox (1987) in describing this situation as closely paralleling the design of a printed circuit board. The electrical paths on the board itself are the interfaces between integrated circuit (IC) chips. The reusable software model objects are the analog of the hardware IC. They can be used in any model framework ("software" printed circuit board) where the interfaces, or "software pin-outs," allow them to be "plugged in." The IMDE will provide an architecture that supports this reusable parts concept by providing stand-alone elements of code which can be "plugged" into a simulation model.

On-Line Help

The LCOM user manual in particular was examined for its capability to help the user learn about the functionality of the system. As a study manual, the LCOM user manual is very good, describing the process of modeling in excruciating detail. However, it does not provide a very convenient quick-access guide to specific questions. Error messages are very cryptic, especially to the novice user. Part of the problem is a direct limitation of the user interface, which currently cannot use graphics or windowing methods to show a more complete and understandable description of the error and how to fix it. An on-line help system to describe menu option functionality, errors, and tutorial information should result in a much faster ramp-up from novice to proficient user. The IMDE will include this type of help facility.

User Interface

The current user interfaces for LCOM and TSAR do little to enhance the modeler's productivity. None of the above characteristics are tied together in a way that allows users of the system to get away from the low-level details and concentrate exclusively on the processes and entities being modeled. No interface can remove all the complexity involved, but many techniques are available to improve the current system. One major step is the integration of all parts of the simulation process into a single user interface environment, thereby building in aspects of configuration control and data formats necessary to construct analysis and report generators as well as the reusable class/object repository. The windowing capabilities provided by current graphical user interface standards provide the user the ability to display a large amount of information on several different windows, and maintain easy access to each one without the menu-backtracking problems that are evident in earlier attempts at graphical interfaces. IMDE will use the latest standards in graphical user interface technology.

III. SYSTEM REQUIREMENTS BASELINE

To achieve all the aforementioned desirable system characteristics, a (large) set of system requirements has been established. Many of these were levied from the Statement of Work (SOW) prior to TASC's assessment of the detailed requirements. For example, the Air Force had already determined that IMDE would greatly benefit from use of object-oriented programming languages and databases. Object-oriented technology, specifically the encapsulation of simulation entities, is a key to the feasibility of IMDE as an environment for the whole simulation process, from model conception through study recommendations. One task was to perform a survey of the available products and select the products to be used based on the findings of this survey. As in any requirements analysis, tradeoffs had to be made between using in-house developed tools and commercially available ones. The survey also investigated the apparent maturity and stability of the products.

Four User Levels

Since the early stages of the contract, the concept of IMDE has included the specification of four different user levels corresponding to different levels of computer sophistication. The Utility/Programmer level allows the most flexibility and requires the highest degree of computer sophistication. At this level, the user has access to the IMDE source code and will be doing a lot of actual Modsim II coding of simulation classes. At the Developer level, model parts (i.e., simulation classes) and algorithms can be defined through graphical tools. The Analyst will be able to take model parts created by the Developer or Programmer and combine them to form complete compiled simulation models. The End User, who requires the least amount of computer expertise, will be able to run different experiments on a compiled simulation model.

Environment Architecture

From the outset it was clear that the OODBMS would play a critical role in the way the environment would work. In fact, three seperate databases are used in the IMDE. They are (1) the Assistance/Help database, (2) the Premodel database, and (3) the Project database. The Assistance database is necessary as a teaching and reference component of the system, but the Premodel and Project databases are the cornerstones of the entire IMDE. Figure 1 illustrates the relationship of the Premodel and Project databases. The Project database stores completed simulation models and their associated input data, output data, and documentation. The Project database is therefore the component of the IMDE architecture that provides the configuration control of completed models. The Premodel database contains parts of a model, i.e., simulation classes. In the airbase logistics domain, examples of parts are individual airbase entities such as aircraft, line replaceable units (LRUs), maintenance squadrons, fuel operations, etc. These parts will be available for use with any model.

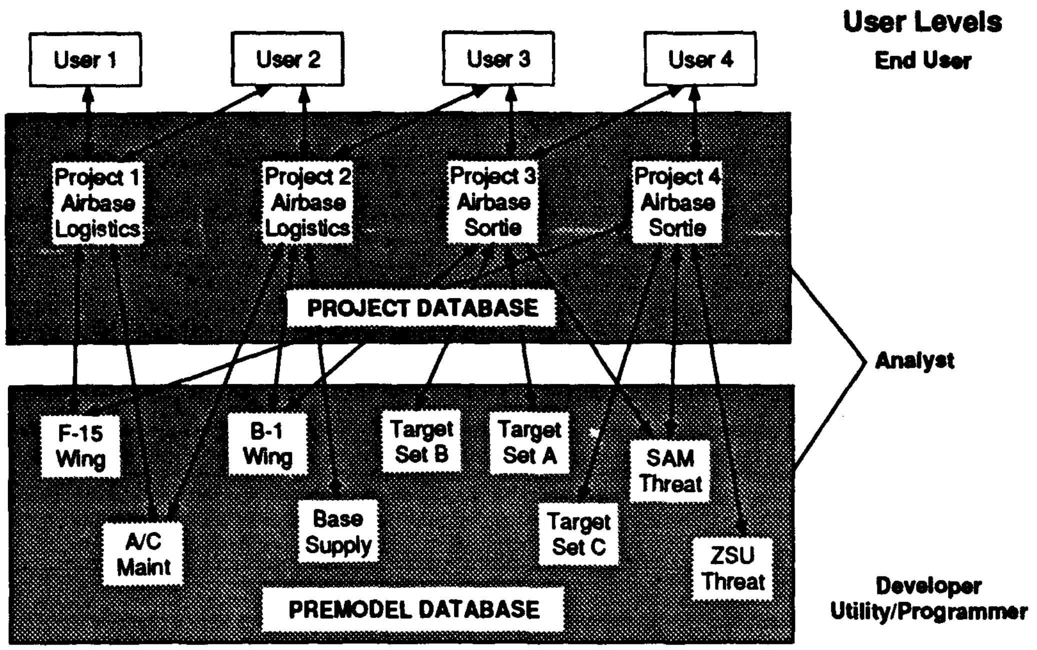


Figure 1. Database Relationships.

Model parts can be used either as-is or with modifications. For example, one modeler might construct a logistics simulation Project using an F-15 aircraft class he created in the Premodel database. At some later point in time, another modeler may want to create another variation on the F-15 logistics simulation. In doing so, he could take the F-15 class created by the first modeler and use it without modification. A third modeler may want to create an airbase operations simulation to look at the effects of aircraft sorties in terms of enemy damage as opposed to depletion of friendly resources. He may well use the same F-15 class as the starting point for the aircraft class in his simulation, but because of his different emphasis, he may modify it to suit a more operational (as opposed to logistical) model of an F-15. The modified classes are stored in the Premodel database in exactly the same way as the original classes, so they can be reused in the same manner.

Figure 1 also indicates which user levels are involved with the different databases. The highest levels of modeling sophistication are required at the Developer and Utility Programmer level. These levels construct the framework for the simulation (the software analog to the printed circuit boards discussed earlier) as well as populate the Premodel database with simulation classes to "plug in" to the framework. Users with significantly less expertise in programming and modeling will be able to work at the Analyst level to choose Premodel database parts to construct their Project and set up input and statistical collection templates for it. After the Analyst has constructed the Project, the End User may run simulation experiments without any knowledge of the Premodel database classes that were put together to form the Project. The End User concentrates on the parameterization of different values set up in an input template created by the Analyst. He changes values of variables of interest, runs the simulator, and analyzes the results.

Preprocessor

Although the OODBMS is the cornerstone of the IMDE, it is not directly visible to the typical modeler. The Preprocessor component of the IMDE architecture encompasses the set of functions which interact with both the Premodel and Project databases to build simulation models. In the case of the Premodel database, the Preprocessor provides for the creation, modification, deletion, and inheritance of model parts.[1] The variables and algorithms of each of the classes are manipulated at this level of model building. The Project Preprocessor provides the functionality to select parts from the Premodel database to create new Projects. It also allows

[1]Inheritance is the object-oriented programming concept of using the definition of a general type of class as the basis for the definition of a more specialized type. For example, an F-15 class could be defined by inheriting all the characteristics of a more general aircraft class, and the designer would only have to respecify the uniquely F-15 characteristics and actions.

for creating new input sets, adding Project documentation, and copying an existing Project template to use as the basis for a new Project.

The Preprocessor represents the model building portion of the IMDE. Most of the user-level specific functionality is contained in this component of the architecture.

Simulator

The Simulator is the component of the architecture that actually causes a specific model to execute and generate raw data. Although the internal functioning of the model being executed can be arbitrarily complex, the Simulator component which starts its execution process is fairly simple. After the user identifies which simulation executable program and input are desired, the Simulator extracts both from the Project database, puts them into the required file format, and runs the executable file, which reads the input file during the course of the simulation initialization. At this point, the executable simulation runs as a stand-alone Unix process. The raw output it creates is saved to a temporary file. When the run is complete, the Simulator stores the contents of the raw data file in the Project database, linked to the specific input record used to create it. In this way, configuration tracking is maintained between the input and output of any given simulation experiment.

Postprocessor

After data from a simulation has been generated in the Simulator and stored in the Project database, the Postprocessor module will be used to construct a meaningful set of tabular and graphical statistics that can be used as an aid to interpretating results, planning future simulations, and/or documenting simulation results and recommendations. The Postprocessor will make it easier to perform proper statistical tests, encouraging novices and experts alike to use them. Having the tools close at hand in an integrated environment will make generating relevant statistics an easy menu selection rather than a tedious data reduction effort.

Standard descriptive statistics generation will be provided in the Postprocessor. In addition, confidence intervals can be generated, possibly after removing outliers and transforming the data to approximate normal distributions. Hypothesis testing will be available to test whether the perturbation of an output variable, due to the variation of one or two input factors, is significant at specified confidence levels. Sensitivity analysis will also be an important tool, allowing a traversal of a range of levels of an input factor to be limited by a specified amount of variation in the output variable of interest. Sensitivity analysis permits a wide coverage of parameterizations by invoking the Simulator multiple times.

Kernel Model Development Environment

The Kernel Model Development Environment (KMDE), or Kernel, is the core of the IMDE that ties together all the other components. It includes the user interface, database access functions, and system administration capabilities such as adding or modifying users or work groups, modifying database object protections, and setting up user environment default values for window placement, color, etc.

IV. MODEL DEVELOPMENT AND MAINTENANCE

This section describes the process by which a simulation model is constructed using IMDE. This example assumes a very simple model having only three classes and that the Premodel database already contains two of those classes. The two classes already created are (1) an avionics maintenance specialist, and (2) a computer control unit (CCU). The class to be created is the Avionics Control Computer (ACC). The CCU is a piece of test equipment that connects to the Automatic Ground Equipment (AGE) connectors on the ACC. The ACC is a general purpose avionics computer LRU. The CCU can be connected to the ACC while it is still on the aircraft to determine whether it has failed. The ACC has a particularly high rate of "could not duplicate" maintenance actions, since the software resident in the LRU is often in error, and the hardware is not really in need of repair. Hence this on-equipment test is important to avoid a large number of unnecessary removals. Each Premodel database class can have attributes which describe its state and methods which describe its possible actions or behaviors. A very simple model is being assumed in this example for the ACC, giving it only the following minimal set of attributes and methods:

Attribute:

fault_detected of type boolean

Methods:

remove_from_aircraft
set_fault
flight_line_maintenance

The attribute "fault_detected" is set to true by "set_fault" if the "on_equipment_test" method of the CCU class detects a problem with the ACC. The "remove_from_aircraft" method is simply modeled as a waiting period of thirty minutes. The "flight_line_maintenance" method requests a CCU and avionics technician, waiting until each is available. Next it calls the "on_equipment_test" method of CCU, waits for it to finish, and releases the CCU since it is available for use again. If "fault_detected" has been set, "flight_line_maintenance" calls the "remove_from_aircraft" method, waits for it to complete, and then releases the avionics technician. If "fault_detected" is not set, the avionics technician is released immediately.

The process just described is greatly simplified from what may need to be modeled in a real-life study. In a realistic example, the magnitude of the model configuration control problem would be more readily apparent. Each part would have many methods, each aircraft would have hundreds of parts, and each airbase would have many other types of entities besides aircraft. The OODBMS can perform the configuration control with ease whereas doing it by hand is very difficult and time consuming. (Configuration control is currently performed by hand.)

To create the ACC, the modeler proceeds to the Premodel Preprocessor screen of the IMDE graphical interface (Figure 2), clicks on the Create button using the left mouse button, and inserts "ACC" into the Class field of the Premodel Class Editor screen (Figure 3). (Note: All screens described here are pictured in the IMDE Software Requirements Specification.) Within the Premodel Class Editor, the modeler can then enter a description for the class or request help.

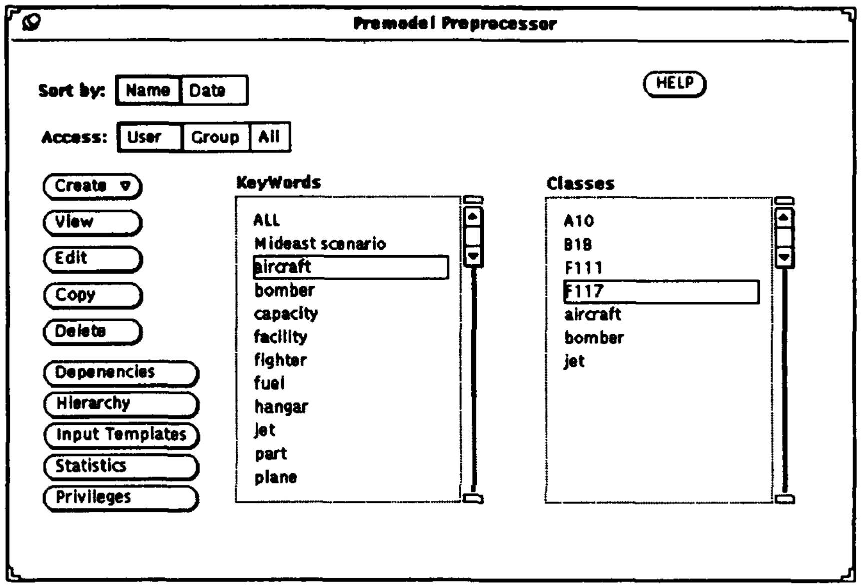
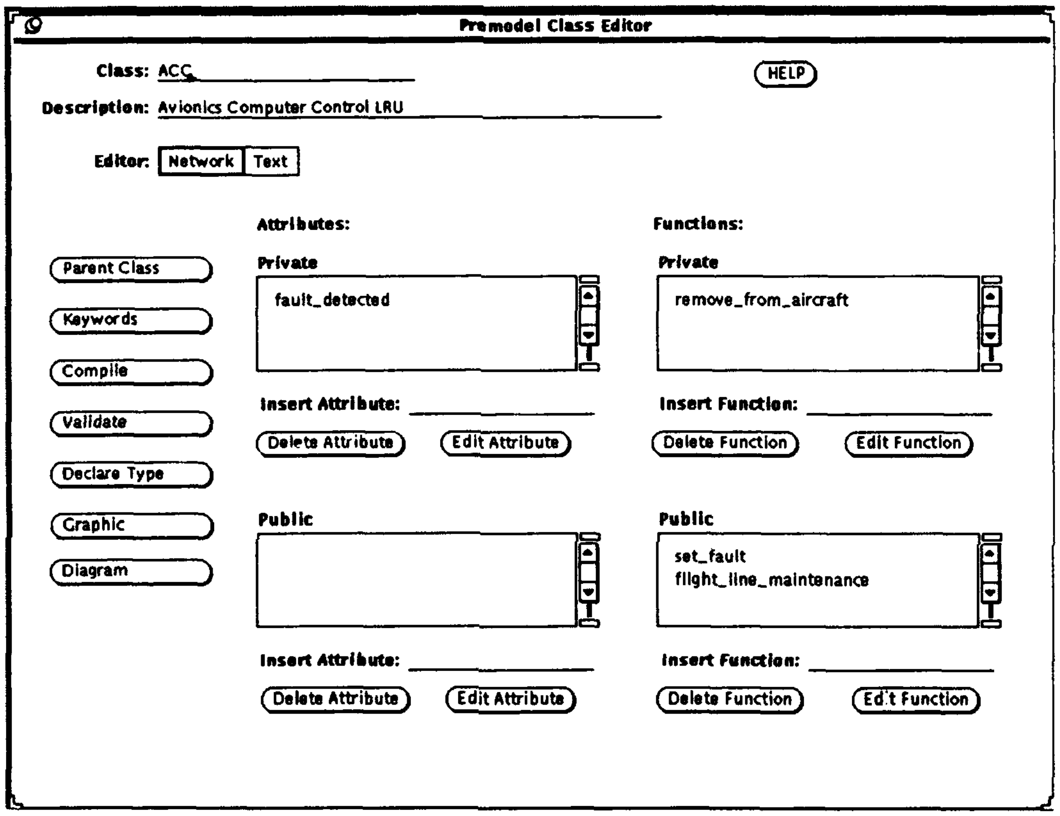


Figure 2. Premodel Preprocessor.

Creating the ACC class requires that its attributes and methods be described via the Premodel Class Editor. Based on a user survey, the term "function" was found to be preferred to "method," so the windows reflect this nomenclature. First the attribute "fault_detected" is entered into the scrolling window for attributes labeled "Private." A private attribute is one that should not be available for use outside the class in which it is defined. Clicking on the attribute just entered will open a window which allows specification of that attribute in terms of type (integer, floating point, string, etc.), default value, whether to hard code the variable or allow it to be selected as an input parameter for variation at the End User level, and whether to allow statistics collection for the attribute. After entering the afigure 5ttribute parameters the modeler can likewise enter the names of the methods in the appropriate scrolling windows. "Set_fault" and "flight_line_maintenance" are public methods since they are called from outside the ACC class. "Remove_from_aircraft" is private since it is called by "flight_line_maintenance," which is within ACC. (In a more complex simulation, "remove_from_aircraft" might be defined as public, so that someone could remove the ACC without going through the troubleshooting steps. This might be done if the ACC was being cannibalized.



Class: ACC
Description: Avionics Computer Control LRU
Editor: Network Text

Attributes:
Private
fault_detected
Public

Functions:
Private
remove_from_aircraft
Public
set_fault
flight_line_maintenance

Figure 3. Premodel Class Editor.

Once the attributes and methods (functions) have been specified, the Developer can design the algorithms for each of the methods defined in paragraph form above. He can do this by selecting the text editor, which allows the programmer to write Modsim II source code. Alternatively, the Developer can select the network editor to design the algorithms in terms of graphical activity networks. It consists of a palette of thirteen nodes corresponding to primitive simulation constructs, which can be dragged onto a canvas and connected by arrows to form a chronological sequence. (The Software Requirements Specification contains a detailed description of each node.) Each node on the palette has a set of parameters which define it.

To enter this set of parameters, the modeler clicks on the node with the right button; this brings up a pop-up window. For instance, the Allocate node allows entry of a node name, type of resource desired, quantity to be requested, and various other options. Nodes can be moved around on-screen to make space for new nodes by depressing the middle mouse button and moving to the desired location. Node connections are maintained during this operation. The canvas can be scrolled horizontally or vertically to provide a node definition area several times the size of the physical screen. A zoom capability is planned in the future.

Networks can be uncomplicated, consisting of only one node, or can be complex. The network for "set_fault" consists of a single "Assign" node which will set "fault_detected" to true. Similarly, the "remove_from_aircraft" method consists of only a "Duration" node, which in this case causes thirty minutes of simulated time to elapse.

The activity network for the "flight_line_maintenance" method is more complicated, and is the network pictured in Figure 4. The first two nodes are "Allocate" nodes, and ask for the CCU and avionics technician. The next node is a "Wait For" node, which waits for the CCU to execute its "on_aircraft_test" method. At this point, the CCU is no longer needed and a "Deallocate" node is used to return it to a pool of available resources. The "Branch" node allows branching based on a condition entered in its parameter window, in this case a simple test of whether "fault_detected" has been set true. If it has, "remove_from_aircraft" is called and execution of "flight_line_maintenance" waits for it to complete before continuing with the next node, which in this case is a Merge. The "Merge" node signals a recombination of separate threads of control. For this example, one of those paths elapsed no simulation time, the other elapsed thirty minutes of removal time. In both cases, the next and last action is to release the avionics technician using a "Deallocate" node.

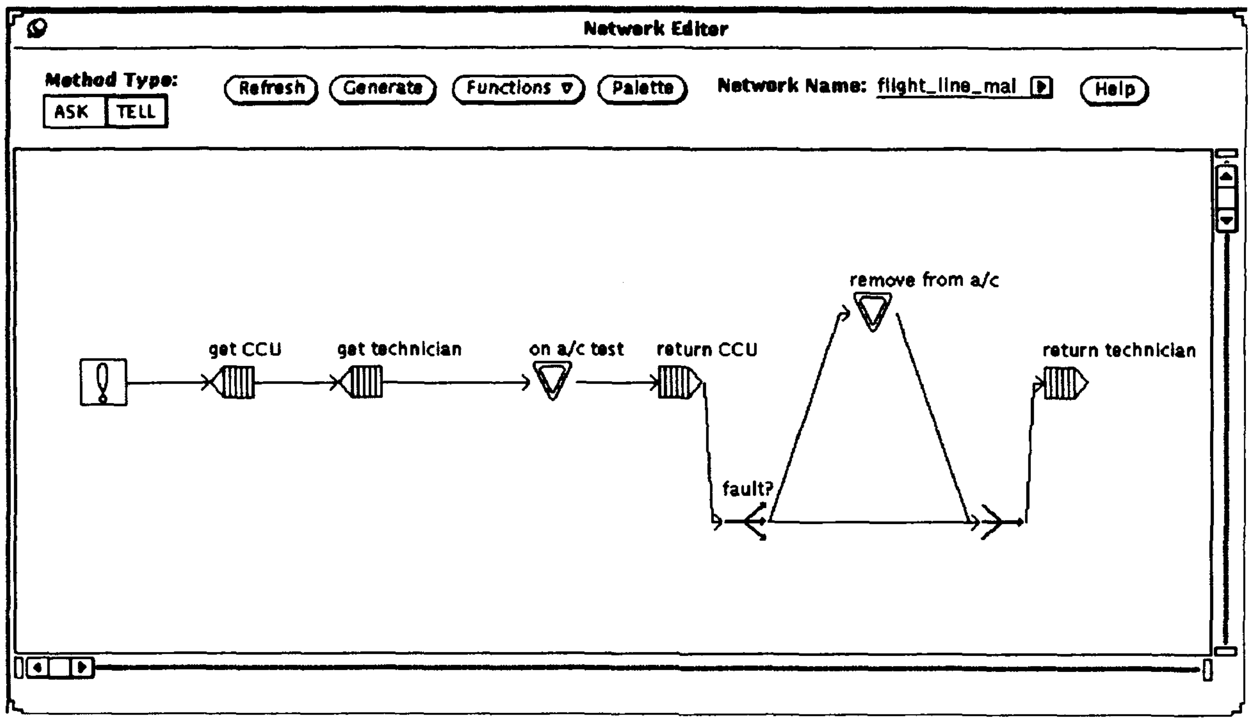
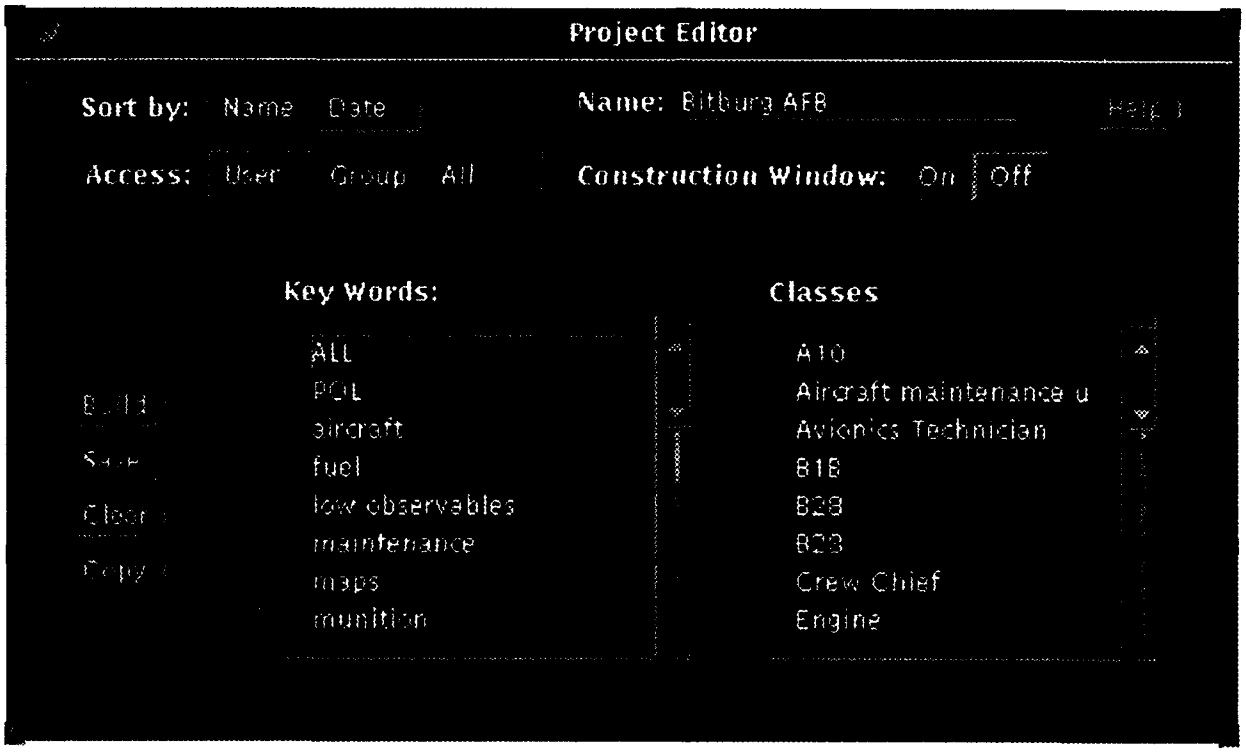


Figure 4. Network Editor.

A network can be saved at any time, and when it is completed the modeler can generate Modsim II source code from the graphical representation automatically by clicking on the Generate button. The source code from the methods is combined with the attribute declarations to form a complete source code representation of the ACC. This part can then be compiled into an object code representation suitable for linking into a Project. Source and object code representations are stored in the Premodel database along with the graphical networks. Other options available at the Premodel Class Editor include defining which variables will generate raw statistics and defining possible input templates for this class. Once those additional steps have been done, the third and final class (the other two were assumed to be pre-existing) is completed and the Analyst is ready to construct the Project from the three parts.

From the Project Editor screen (Figure 5), the Analyst is able to select desired classes by clicking on them in the scrolling Classes window. If desired, he can select an input set and a statistics set for each of the three classes that were selected to be in the Project (each class may have several of each). He can then "Build" the Project, which links the object code and statistics modules of the classes that were selected, forms the total input record for the Project as the sum of chosen input records from the constituent classes, and performs some bookkeeping functions. At this point, given a successful build, the Project is ready for execution in the Simulator.


Project Editor

Sort by: Name [✓ Date]   Name: Bitburg AFB   Help

Access: [✓ User] Group All   Construction Window: On [✓ Off]

Build
Save
Clear
Copy

Key Words:	Classes
ALL	A10
POL	Aircraft maintenance u
aircraft	Avionics Technician
fuel	B1B
low observables	B2B
maintenance	B23
maps	Crew Chief
munition	Engine

Figure 5. Project Editor.

It should be apparent that there is still a great deal of complexity to simulation, even with a tool like IMDE. However, many aspects are much improved, notably the capability to browse through a database of existing indexed Projects or Premodel classes by keyword to find a simulation or model algorithm that has not been used lately. The capability for nonprogrammers to examine the algorithms in models allows more people to participate in the simulation model verification effort. This actually goes beyond just helping to maintain models. It allows a much larger group of analysts to develop and use simulation models, since IMDE requires less programming expertise.

V. POTENTIAL DATA SOURCES AND MAINTENANCE PROCEDURES

The eventual success of any logistics modeling tool is very dependent on its ability to automatically load applicable portions of the existing maintenance databases, and to keep that information current. These tasks are not currently within the scope of the IMDE SOW, but are very important in order to avoid the necessity of manually entering extensive amounts of data on each of the modeled weapon systems.

Potential Data Sources

There are several sources of Air Force maintenance and logistics data which could be accessed to obtain inputs for a simulation model. Some of the data systems are described here, but more research is necessary to identify other applicable data sources and to narrow the selection down to specific data fields. Since IMDE is a modeling environment as opposed to a specific logistics model, providing a flexible link to databases will be very important.

D056 is a Product Performance System which provides reliability and maintainability data on weapon systems and subsystems. D056 is scheduled to be absorbed into the Logistics Management Systems (LMS) Modernization Program, established by Air Force Logistics Command (AFLC).

As part of the LMS Modernization Program, the Reliability and Maintainability Information System (REMIS) will become the primary Air Force database for collecting and processing base, depot, and contractor maintenance and inspection information. It is structured by weapon system and major equipment category. REMIS will replace multiple current data systems, consolidating data regarding equipment inventory status and utilization, mission capability and awaiting parts, reliability and maintainability, weapon and support system configuration and modification, and Air Force Specialty Code (AFSC). As the primary source of Air Force Reliability and Maintainability 2000 data, the system will aggregate all product performance data by fleet, mission design, series, end item, and component.

The Recoverable Consumption Item Requirements System D041 includes actual and estimated data on item demand rates, condemnation, base processing percent, order and ship times, repair times, unit cost, inventory levels, and item configuration among others. D041 is scheduled for integration with other systems to form a Requirements Data Bank (RDB). RDB will compute material requirements for three major categories of Air Force items: equipment such as trucks and test stands, recoverables such as subassemblies, and consumables such as repair parts and hardware items used in repair. This information is based on the indentured application of each individual part of a particular aircraft or end item and all other applications of that part.

The RDB will interact with all AFLC core logistics functions and the information systems programs that support them. The information exchanges are needed to either compute requirements or execute actions as a result of requirements computations. The primary information deals with demands, pipelines, and inventories as reported from Stock Control and Distribution (SC&D) and flying hours and activity rates as projected by the Air Force through the Planning, Programming, Budgeting System (PPBS). The SC&D system controls the storage, allocation, and movement of AFLC inventories. Tracking quantity, condition, and location of assets, the system receives and sends data concerning asset balances, consumption information, shipping times, and parts requisition status.

The Visibility and Management of Operating and Support Costs (VAMOSC) system is a database system of operating and support costs for Air Force weapon systems and their major components. The data in VAMOSC are collected from over 15 other Air Force data systems. VAMOSC contains two major reporting subsystems: the Component Support Cost Subsystem (CSCS) and the Weapon System Support Cost (WSSC). The CSCS contains costs associated with the major components of each weapon system. These data are tracked by Work Unit Code (WUC) and National Stock Number (NSN). A transaction driven cross-reference file identifies maintenance that has occurred during the reporting period. The WSSC subsystem contains costs associated with the weapon system as a whole.

In particular, WSSC contains data elements on manpower composition for aircrews, mission support personnel, and maintenance. These data arrive from the Base Level Personnel System (BLPS) E300Z aggregated by command and geographic location. Through WSSC, it is possible to obtain visibility into manpower staffing by AFSC. H069S system provides command and base costs for each WSSC cost category. H036C provides costs generated at depots. H033A provides command/base/MDS aviation fuel consumption quantity and costs. G099 provides command/base MDS quantities, and A21 provides command/base MDS flying hours.

Data Maintenance Procedures

Initial uploading of maintenance information into the IMDE databases will provide a base of parametric data on many systems. However, this data may change significantly for given objects over time, which creates a need for periodic updates. This is not as simple as just replacing the old values. When doing a simulation study, a well-defined baseline is needed for the input data. If a set of runs is made on Monday and the database administrator "updates" the database on Tuesday, runs performed on Wednesday with a Project made of the same classes could possibly give different results. A solution to this problem might be to maintain old data sets in parallel with the new ones for a period of time, or perhaps indefinitely. The database schema would have to be designed to include this multiple parameter value record for each object being read from maintenance data.

VI. SYSTEM APPLICATION CONSTRAINTS

The IMDE as currently designed has no real ties to specific simulation domains, although the emphasis for the current contract is to develop a set of domain classes for airbase logistics simulation studies. This means that IMDE users can construct an airbase operations or communications simulation just as easily as an airbase logistics simulation, if given the right classes in the Premodel database. This capability should make IMDE a very popular tool in any complex domain area.

VII. FUTURE DIRECTIONS

The current work toward developing the environment and a demonstration-level set of Premodel airbase classes will continue through March 1993. Additional effort is needed to facilitate the conversion of existing databases with the IMDE. This work is not pushing technology, but it does require extensive coordination with the data generating organizations as well as the coding of the conversion programs.

Another possibility for "extending" IMDE would involve beefing up the set of airbase classes in the Premodel database to make the environment more robust from the Analyst's point of view. For example, designing more detailed aircraft models, base templates, and mission scenarios would give the Analyst a greater variety of model parts to use in building Projects.

Finally, the issue of programming languages has been raised several times during the course of the project to date. Modsim II requires a fairly large capital expenditure for each node in a potential IMDE network, and also makes IMDE dependent upon a nonstandard programming language. A C++ simulation library could be derived from the MCC CSim product, which costs $125 for the source code, and allows unlimited use of any derivative products. Converting to C++ would also provide the advantage of a more direct representation in the Versant database.

Of course, C++ is also not a "standard language" in the DOD sense. The current portion of IMDE written in C++ (the environment) could be coded in ADA in the future, given the availability of X window system packages, which appears likely in the near term. Spectran, TASC's C-to-ADA translation tool, might be useful in such a recoding effort. It could prove more difficult to translate the part of IMDE currently to be targeted for Modsim II into ADA. This part represents the domain classes (for the current contract, airbase logistics classes), which are currently dependent on the object-oriented concepts of inheritance and dynamic binding. These features are not present in the ADA language; however, the proposed ADA 9X standard does include these features. In addition, it would be necessary to develop or package an ADA simulation package.

VIII. CONCLUSIONS

The work performed to date clearly indicates the feasibility of the IMDE concept. A fairly complete user interface has been constructed to encompass the entire simulation process. Just as importantly, all of the key technologies required to fill out the functionality behind the user interface have been demonstrated. The work that remains under the contract is to implement the design as defined in the Software Requirements Specification. To obtain the "operational" capability of LCOM, IMDE still requires (1) an expansion of the Modsim II coding effort to include a more complete set of airbase entities, and (2) the development of automated data feeds from existing Air Force databases.

REFERENCES

Cox, B.J. (1987). Object Oriented Programming: An Evolutionary Approach. Reading, Massachusetts. The Addison-Wesley Publishing Company.

★U.S. Government Printing Office: 1992 — 648-069/60205